Requirements for Code Distribution and File Access at Intensity Frontier Experiments

A.Norman, A.Lyon

Overview

In the current offline data processing model used by the Intensity Frontier (IF) experiments, a combination of experiment specific software libraries and support files be need to be available to offline jobs that are running on either Fermilab or non-Fermilab computing resources. In the current model this access to software libraries and support files is provided through a centralized storage system (Bluearc) which serves as the home for the experiment's software distributions and associated files. This central storage system is made available through a standard set of mount points that are exported to computing nodes at Fermilab.

Offsite computing facilities need to rely on a separate installation of the experiment's software that is accessible to their computing resources. These offsite installations require not only storage resource but site librarians to maintain the software distributions and keep them synchronized with the rest of the experiment's software.

This document examines the current methods that different intensity frontier experiments are using to handle code distribution under these conditions and the possible applications of new technologies to improve the distribution of experimental code, common libraries, shared data and other resources across a wide variety of computing resources that are now available to the IF experiments for data processing.

In particular, this document examines that impact of an on-demand caching file system like CVMFS and its application to the current IF computing model.

The CVMFS file system was developed by CERN to provide users working with large offline code bases from their desktop or laptop computing, a transparent means of remaining synchronized with the central authoritative software distributions and without the overhead of transferring the entire code base to their personal computer. The CVMFS system accomplished these tasks through the use of a central authoritative distribution server which exports the code base through a web interface and a FUSE (File System in Userspace) module which maps the file access into a Linux filesystem with cache capabilities. The result is a system which acts as an on demand transfer agent (files are only transferred when accessed by the OS) and a local cache improving the efficiency of the system for files that are commonly accessed.

This document is broken into two general parts. The first part describes the different data handling use cases that are currently being used at Fermilab by the

different IF experiments. The second part describes the common requirements that these use cases impose upon a potential deployment of an on-demand caching filesystem service, like CVMFS, for Fermilab experiments.

Experimental Use Case (Current)

There are currently four major experiments that are part of the Intensity Frontier program. Each experiment is currently performing data analysis using either the FermiGrid facilities or other computing resources at Fermilab and have code or data access patterns that are candidates for being handled with CVMFS. In this section we describe the current operation of these experiments and then discuss the portions of their computing model that would potentially benefit from CVMFS.

MINOS Data Handling

The Minos experiment has been doing large scale batch processing on Fermigrid resources for data processing and will continue to use these resources for the fore seeable future. In the current offline processing model the Minos offline jobs access the majority of their code, libraries and support files directly from the central disk service. In particular minos jobs:

1. Run the binary executable for all jobs directly from the bluearc disk service. This include loading the shared libraries and other binary files required at startup directly from the bluearc disk. This loading of job binaries and shared libraries is small and estimated to be in the hundreds of megabytes to 1 gigabyte range for most jobs. The issues associated with simultaneous access to the job binaries and shared libraries during the submission of large numbers of jobs are avoided by a staggered startup. This staggered startup occurs to due to restrictions that are placed on copying the datafiles for individual jobs to disk that is local to the node on which the job is running. These copy operations are protected by a concurrent access mechanism (cpn copy wrapper) which then spaces out the job starts based on the copy locks on the datafiles.
2. Require access to large event template libraries. These template libraries are referenced during the job and are common across all jobs of a given class. The file sizes are on the order of 3+ GB and are static across offline releases. These files are accessed directly from the Bluearc but can be transported with some jobs to the local scratch disk of the node the job is running on. The impact of the transfer (if it is transferring to local scratch), because it is performed with a “cpn” access wrapper which essentially serializes job startup and provides extra contention for bluearc access with the data file transfers, becomes a significant source of overhead especially when being used on jobs with short overall data processing run times.
3. Require a number of smaller auxiliary files for calibrations and other conditions information that are loaded directly from bluearc. These files are

small enough and are accessed infrequently enough or a single time at job startup, that they provide little impact in overall offline processing.

Minos currently performs data processing solely at FNAL, but does perform significant amounts of Monte Carlo event generation at sites other than FNAL. This task provides a different set of constraints based primarily on the ability of the remote site to not only deliver the job libraries and auxilary files, but to stay synchronized with the master code releases.

In the current Minos offsite Monte Carlo system, five sites:

1. Caltech
2. Tufts
3. Rutherford Lab
4. The College of William & Mary
5. University of Texas, Austin

generate Monte Carlo for the experiment. Each site is administered separately by an individual associated with the site and executed only on resources that have been identified by that site for Minos MC production (i.e. production does not occur through an OSG batch system). The code for each site is kept in sync with the primary Minos code base at FNAL through the manual distribution of analysis suite. In this model, each site librarian receives the base tar-ball with the experiment code and required flux/cross section files. The code is built by the local librarian and installed on their clusters in a manner that can be unique to the local cluster. Flux and cross section files having sizes in the 10's of gigabyte ranges are installed at the remote site, but a method of distributing them to each job at run time must still be handled according to the local site's infrastructure (i.e. central disk serverices at the remote site). In addition access to data that is stored Minos databases must be arranged and can be unique to the site.

Nova Data Handling

The Nova computing model relies heavily on use of the central Bluearc disk system. This reliance on the central disk currently prevents the Nova collaboration from running large scale data processing on resources other than the FermiGrid and other clusters at FNAL which have the Bluearc visible to them. The Nova computing jobs current load the initial code base from central disk, load the executables, supporting libraries from central disk, performs UPS setups of external products from central disk. Each job that runs on a worker node additionally needs access to a set of large [~GB size] Monte Carlo flux files, cosmic overlay files [~ GB for far detector cosmics], bad channel masks [100's MB], beam quality files, and the geometry files (translated into root format) as well as small number of additional support files. All of these are loaded directly from the bluearc and no attempt is made to throttle the file accesses. (Note: this has caused problems in the past since accessing some of the larger geometry, bad channel maps and cosmics files can overload the bluearc.)

Because the Nova uses the ART framework, the jobs also need to have access to the the full suite of ART external products and support files, as well as to the library of FCL configuration files that is located under both the experiment's code release structure as well as under the ART distribution (i.e. user vs. standard framework configs)

The search paths that are setup for all these files are set though a combination of the Soft Rel. Tools (SRT) code management system, the ART configuration language (FCL), the Unix Products Setup (UPS) system and the job submission system. This means that the actual location that is used for a given file may not be determined until run time and may in some cases vary depending on which analysis modules are loaded during the event processing. This highly dynamic configuration and on demand loading system make it difficult a priori determine all the file dependencies that a single job may have, preventing some common methods of file distribution such as static binary builds or the distribution of a small set of runtime libraries and config files with the base job. Instead the only safe method of distributing the code base is that every job must have access to the entire NOvA distribution and its externals.

[NOTE: It may be possible to install a relocatable UPS distribution on a caching file system like CVMFS. The impact of doing this would be a type of on-demand UPD. The file system would have to be setup with a long (or infinite) cache time for the UPS distribution. Then you would export the entire UPS tree. When a product is first setup and the file actually accessed, the initial transfer of the files would occur to the local disk and cause a long latency prior to access (it may also fail depending on how the underlying file systems blocks or times out). All subsequent accesses would find a cache hit and the product would instantly be available as if it were installed locally. The advantage of this system is that you would obtain the strong versioning and flavor tagging of the UPS system which would allow you to run across multiple platforms in a semitransparent manner (i.e. your UPS distribution for the grid would include the appropriate flavor builds to permit use of whatever hardware your job landed on, x86_64, i386 etc…. and you would only actually ever transfer the appropriate flavor to that node.)]

Minerva Data Handling

The Minerva computing model is very similar to the Nova model. The base jobs (executables and libraries) are run from the central bluearc disk. This core code distribution encompasses approximately 5GB of files split over the gouty framework and user libraries. In addition a limited number of small auxiliary files are required for each job which have a size that is typically a few 100's of megabytes. These files are common across job classes (i.e. Monte Carlo jobs vs data processing jobs) and represent condition/calibration style data that is used by the job. The raw data for each job is copied directly from the central disk to the local node where the job is running. This copy procedure uses the standard cpn utility to limit the simultaneous accesses to the central disk.

MiniBooNE Data Handling

MiniBooNE currently uses only onsite computing resource for both Monte Carlo generation and for data processing.

MiniBooNE jobs are run primarily from a private cluster of approximately 60 compute nodes. On these nodes the MiniBooNE base release is homed in an AFS area and jobs are run alive off of the AFS system. In addition, MiniBooNE jobs can be run from grid resources at Fermilab. For grid jobs the code/support libraries for the base release exist separately on the /grid/app area of the Bluearc central storage. This area is available to each worker node that a jobs runs on as an NFS mount.

At the start of an analysis job these libraries and support files need to be accessed and loaded by the local system. The amount of data that needs to be to transferred per MiniBooNE job varies from between 150MB to approximately 1GB depending on the job type. For general grid jobs this code distribution is performed via a gridftp stage which transfer the required job and data files to the working directory on the compute node and then stages out the output files at the end of the job. [Note: C.Polly notes that the code distribution is homed on the Bluearc but transferred with gridftp. Check on this since this seems like a more complicated way of performing the transfer than just using a cp tool to cache the files from the Bluearc to the local node.]

In addition to the analysis libraries and datafiles, all MiniBooNE jobs require access to a copy of the boone database (BooDB) which is currently resident on the Bluearc central storage under the /grid/fermiapp hierarchy (is this also on AFS?). Jobs access this file a at least once during their life cycle to retrieve calibration and conditions data. The file was moved to reside on central storage because it was found that accessing the information directly from the database server would overload the server during periods of intense batch processing. The current access method avoids directly querying the DB server at the cost accessing the central storage device.

Impact of CMVMFS

The MiniBooNE experiment could potentially benefit from the development of a CMVMFS system in the following ways:

1. Instead of multiple base release distributions only a single code distribution will need to be maintained (and would not be tied to AFS)

1. The shared [static] database file (BooDB)

Mu2e Data Handling

The Mu2e experiment currently uses an online and simulation environment based off the ART analysis framework. Because Mu2e uses the common ART framework, externals and support libraries, their computing model is very similar to

that of the NOvA experiment, where the code base, simulation libraries, and auxiliary files are homed on the Bluearc central file service and made available to jobs running on FermiGrid resources via the common bluearc mounts.

Commonality in File Usage

The IF experiments share a number of common designs in their data processing models. These commonalities can be grouped by the manner in which they are accessed by jobs. The first set of files are those which are used across ALL jobs of a given job class (production, analysis, Monte Carlo) and repeatedly accessed either as a whole or in part when jobs run. These core job files should be distinguished from data files which are consumed by each job and as a result are never common across multiple jobs (i.e. one and only one job in a single submission will ever analyze a given run/subrun file)

Core Files

1. Need for distribution of the base analysis framework executables and libraries. These libraries and binary executables have typical sizes of ~1GB.
2. Need for distribution of the external packages and libraries that the frameworks rely on. These externals have sizes on the order of 5-40GB.
3. Need for distribution of experiment specific software. The typical size of an experiment's code base/libraries is 2-5GB.
4. Need for distribution of experiment specific common block data files (i.e. flux files, static database files, calibration files) These files typically have sizes ranging from 100's of MB to ~GB.
5. Need for distribution of experiment specific configuration files and small utility files. These files are typically tiny (KB to MB range) but they are likely to change on a frequent basis or to have new versions developed that need to be managed in a given tree.
6. Need for distribution of small but numerous conditions/calibration files (e.g. Minos style beam files or Nova style channel bad channel maps and calibrations. These files have sizes of ~10MB but there can be thousands of them tied either to the run, subrun or time period during which a run was taken.

Consumed Files

1. Need for delivery of raw data input files, or Monte Carlo input files from a previous stage of processing. These files have typical sizes ~500MB-2GB.

The access patterns on these types of files differ dramatically. In the case of the consumed files, these are almost always subdivided into "event" blocks and an analysis job will almost always sequentially consume the file, reading either all the events in the file or some subset of the events. This presents the need for the file to be transferred in its entirety to local disk to allow for the job to scan through it.

The properties of all these types of files have been summarized in Tables 1 and Table 2. Table 1 describes the properties of the files as related to their storage. These file properties are ones the affect primarily the delivery of the file between the source and the worker node on which analysis or simulation jobs are run. The

size characteristic relates to the total estimated size of the files within a given class. As an example, for the job libraries category a single library may have a size less than a gigabyte but the collection of all libraries linked to a given job and that must be made available to the job at run time would have an expect total size of 1-6GB. Similarly the number of files field relates to the total number of files in a given class that must be made available during the lifetime of a job. In this case the order of magnitude of the file class is given. By way of example the full set of all external packages that must be made available to a job can have on the order of thousands of total files that need to be installed or made visible to the job site, even though only a small handful of those are actually used at run time (e.g. the typical ROOT or BOOST distributions contain not only the run time libraries but the additional source and header files which can number in the thousands). For the update frequency field we have identified four man classes of files. Many files that fall under the external packages or experiment code repositories have a well defined tagged or versioned management structure and while they can change, do so only as part of a update to a larger release. In contrast, individual users have code bases that change rapidly or are not subject to a larger release structure. For support files, many experiments use some type of block data (flux files, cross section tables) or database snap shots to provide access to well established common information that is required for processing. These database snapshots and large block data files are extremely static and may not change across many releases of the experiment's core analysis code. In contrast other types of auxiliary files, primarily calibration related information, is not only used by a job or set of jobs but actually updated by it. This class of files is highly volatile and presents a challenge for distribution.

Table 1 Properties of different catagories of files requried for Intensity Frontier data processing.

File Type	Size Range	Number of files	Update Freq.	Current Delivery	Cachable
Job Binaries	0.5-2GB	O(1)	Users/Release	Bluearc	X
Job Libraries	1-6GB	$O(10^2)$	Managed Release	Bluearc	X
External Packages	10-40GB	$O(10^3)$	Managed Release	Bluearc	X
Aux. Data	1-6GB	$O(1\text{-}10^2)$	Managed Release	Bluearc	X
Config Files	kB-MB	$O(10^2)$	User/Release	Bluearc	X
Data Files	1GB	1+ per job	Static	Bluearc/SAM	No
Database Snapshot files	<100MB	O(10)	Static	AFS	X
Calib. Database Files	<100MB	O(10)	Volatile	Bluearc/AFS	??

In addition to the physical properties of the files that drive many of the requirements, the manner in which the file is accessed or consumed drives additional requirements. Table 2 lists the same major classifications of files along with their current or expected use pattern. For the access pattern field we have distinguished between files which typically require only a single read access, versus those which may perform multiple but infrequent accesses to a file, versus those which are repeatedly or continuously accessed by the analysis job. The access time field describes the most common times at which the given files need to be accessed. For many of the core code files and libraries these accesses occur only at job start when the binary images are loaded into memory and execution is started. In contrast some files are accessed throughout the entire lifetime of the job, while others are only required at specific boundaries within the jobs. The last fields in Table 2 are the fractions of the files that are typically accessed and the manner in which that access occurs.

Table 2 Access patterns used for different catagories of files used in Intensity Frontier data processing.

File Type	Access Pattern	Access Time	Fraction Accessed	Access Method
Job Binaries	Single	Job start	All	Sequential
Job Libraries	Single	Job start	Sparse	Sequential
External Packages	Single	Job start	Very Sparse	Random
Aux. Data	Multiple/Continuous	Job lifetime	Variable	Format Dependent
Config Files	Multiple	Job start and re-init	All	Sequential
Data Files	Continuous	Job lifetime	All	Sequential
Database Snapshot files	Continuous	Job lifetime	Variable	Random
Calib. Database Files	Multiple/Continuous	Run boundaries or each event	Variable	Random

Common Requirements for a File Distribution System

The current use cases, file properties and access patterns described in the preceding sections impose the following requirements on a system that is used to distribute experimental code, libraries, configuration files and auxiliary block data files.

Simultaneous File Access Requirements

- The file distribution system must support on demand distribution of at least 10^4 individual files per experimental code release.
- The file distribution system must support files with sizes up to at least 8GB.
- 1-2GB of Program binaries must be available at run time to the local batch node and the file distribution system must support simultaneous startup of 500-1000 jobs accessing the same core binaries. Alternatively the batch or file distribution system must support a method of staggering the start of jobs or file distribution in time to reduce the simultaneous access to the program binaries.
- Experiment specific and external product libraries must be available at run time to the local batch node as well as throughout the life of the job (to support on-demand loading of analysis modules). The file distribution system must support simultaneous startup of 500-1000 jobs for library loading up to 6GB of binary data, and must support loading and reloading of additional libraries during the life of the job where the typical life of a job is 3-24hrs.
- External product dependencies must be available at run time to the local batch node as well as throughout the life of the job. The file distribution system must support accessing up to 40GB of external product files.
- Auxiliary files with sizes up to 6GB must be simultaneously accessible to all currently running analysis jobs that require the file (i.e. while 500-1000 copies of the *SAME* job may be started at a given time, many more jobs from the same experiment may be running and require access to the same information that is stored in the auxiliary file. Examples of this are flux and cross section files, event template files and detector geometry files which are shared across wide classes of analyses and simulations) for the lifetime of the jobs.
- Static database snapshots with sizes ranging from 100's of MB to 1GB must be made available to all running analysis jobs of a given type.
- Volatile database snapshots with sizes ranging from 100's of MB to 1GB must be available to all running analysis jobs of a given type and if updated the changes must be appropriately updated and kept coherent with the primary distribution (I'm not sure this is compatible with a caching system like CVMFS and we may have to rethink how we deal with items like this)
- If caching systems are employed that make use of local disk on the worker node, the local cache system must support simultaneous access by N running jobs, where N is the number of batch slots that are mapped to the physical worker node (e.g. if an 8 core worker node has 8 batch slots mapped to it, then the cache system must support simultaneous access by at least 8 jobs.)

Management, Updates and File Synchronization Requirements

In addition to the simultaneous access requirements the file delivery system must support the following file management requirements.

- The file distribution system must support cataloging of at least 10^4 individual files with file sizes ranging from a few bytes to 8GB.
- The file distribution system must support the organization of files into well defined releases or versions where individual files in a given release may have the same filename as previous releases.
- The file distribution system must present a file system like directory hierarchy to the analysis job and must support user/group level access permissions.
- The file distribution system must support the distribution of code bases for multiple experiments and support updating of any one experiment without adversely affecting the others
- The file distribution system must have a capability to reference an authoritative central repository to obtain the official copies of experiments analysis distributions
- Updates to the experiment's analysis distributions should propagate to all sites that are using the file distribution system (to prevent version skew between sites)
- Updates should be able to be performed by designated experiment representatives (and not undesignated individuals)
- The file distribution system should support a method of performing incremental updates and bug fixes (i.e. pushing a patch or performing a partial rebuild of a distribution)
- Caching file distribution systems must support a method of preserving cache coherence across all worker nodes that are accessing the files.

Summary

This document explores use cases for program, library, and auxiliary files at the intensity frontier experiments, identifies commonalities among the access patterns, and formulates requirements that a file cache and delivery system should satisfy. Such file access patterns are quite different than those for event data files, as the former are shared among potentially thousands of running jobs. A mechanism for delivering and caching these job files needs to be efficient in both speed and space and able to handle a potentially very large access load.

CERNVM-FS is a potential system that could satisfy the requirements specified here. It transports file images over http via *squid* servers, leading to inherit scalability. Each worker node could be set up to host a CERNVM-FS cache. Only files that are actually used by a job are transported to the cache. This on-demand caching scheme will potentially reduce the total size of all files and images that need to be

transmitted to the client system and will keep the total size of the actual code distributed reasonable.

The additional benefit of the CVMFS caching scheme is that having a shared persistent cache on each worker node alleviates the problem of 1000s of jobs all trying to access the same file on the same volume simultaneously. Instead, the cache needs to be capable of handling only the handful of jobs run concurrently on a single worker node.

The CVMFS system also allows for the definition of centrally managed frozen releases, which can be transparently distributed and updated across all the sites that are subscribed to the CVMFS share. This central management scheme also has the benefit of considerably reducing the burden imposed on site librarians to maintain and update the experiment's.

Much work still needs to be done to characterize performance, reliability, and ease of use of CERNVM-FS, as well as examine the applicability of other delivery/cache systems to satisfying these requirements.